3.1.1. GEOMETRIC OBJECTS

Our fundamental geometric object is a point. In a three dimensional geometric system, a point is a location in space. The only property that a point possesses is that point's location; a mathematical point has neither a size nor a shape.

Points are useful in specifying geometric objects but are not sufficient by themselves. We need real numbers to specifying quantities such as the distance between two points. Real number – and complex numbers, which we will use occasionally – are examples of scalars. Scalars are objects that obey a set of rules that are abstractions of the operations of ordinary arithmetic. Thus, addition and multiplication are defined and obey the usual rules such as commutivity and associativity. Every scalar has multiplicative and additive inverses, which implicitly define subtraction and division.

We need one addition type – the vector – to allow use work with directions. Physicists and mathematicians use the term vector for any quantity with direction and magnitude. Physical quantities, such as velocity and force, are vectors. A vector does not, however, have a fixed location in space.

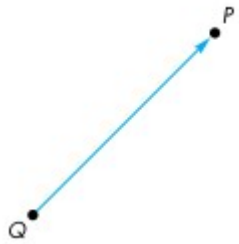

Figure 3.1. Directed line segment that connects points.

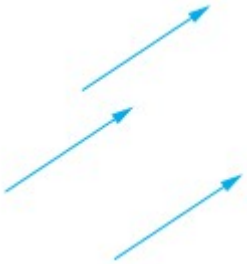

Figure 3.2. Identical vectors.

In computer graphics, we often connect points with directed line segments, as shown in Figure 3.1. A directed line segment has both magnitude – its length – and direction – its orientation – and thus is a vector. Because vectors have no fixed position, the directed line segments shown in Figure 3.2 are identical because they have the same direction and magnitude. We will often use the terms vector and directed line segment synonymously.

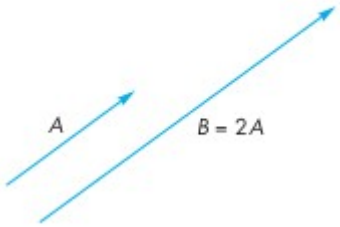

Figure 3.3.1. Parallel line segments.

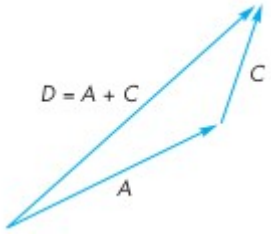

Figure 3.3.2. Addition of line segments.

Vectors can have their lengths altered by real numbers. Thus, in Figure 3.3.1. line segment *A* has the same direction as line segment *B*, but *B* has twice the length that has, so we can write *B=2A*. We can also combine directed line segment by the head-to-tail rule, as shown Figure 3.3.2. Here, we connect the head of vector *A* to the tail of vector *C* to form a new vector *D*, whose magnitude and direction are determined by the line segment from the tail of *A* to the head of *C*. We call this new vector, *D*, the sum of *A* and *C* and write *D = A + C*. Because vectors have no fixed positions, we can move any two vectors as necessary to form their sum graphically. Note that we have described two fundamental operations: the addition of two vectors and the multiplication of a vector by a scalar.

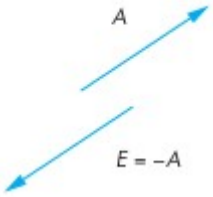

Figure 3.4. Inverse vectors.

If we consider two directed line segments, *A* and *E*, as shown in Figure 3.4, with the same length but opposite directions, their sum as defined by the head-to-tail addition has no length. This sum forms a special vector called the zero vector, which we denote ***0***, that has a magnitude of zero. Because it has no length, the orientation of this vector is undefined. We say that *E* is the inverse of *A* and we can write *E = -A*. Using inverses of vectors, scalar-vector expressions such as *A + 2B – 3C* make sense.

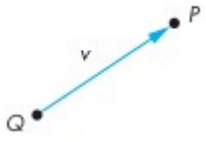

Figure 3.5. Point-vector addition.

Although we can multiply a vector by a scalar to change its length, there are no obvious sensible operations between two points that produce a point. There is, however, an operation between points and directed line segment to move from one point to another. We call this operation point-vector addition, and it produces a new point. We write this operations as $P = Q + v$. We can see that the vector v displaces the point Q to the new location P.

Looking at things slightly differently, any two points define a directed line segment or vector from one point to the second. We call this operations point-point subtraction, and we can write it as $v = P – Q$. Because vectors can be multiplied by scalars, some expressions involving scalars, vectors, and points make sense, such as $P + 3v$, or $2P – Q + 3v$ (because it can be written as $P + (P – Q) + 3v$, a sum of a point and a vector), whereas others, as $P + 3Q – v$, do not.

3.1.2. COORDINATE FREE GEOMETRY

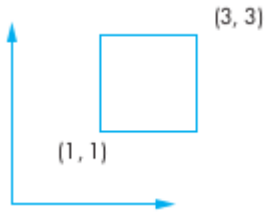

Figure 3.6. Object and coordinate system.

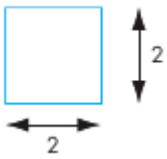

Figure 3.7. Object without coordinate system.

Points exist in space regardless of any reference or coordinate system. Thus, we do not need a coordinate system to specify a point or vector. This fact may seem counter to your experiences, but it is crucial to understanding geometry and how to build graphic systems. Consider the two dimensional example shown in Figure 3.6. Here we see a coordinate system defined by two axes, an origin, and a simple geometric object, a square. We can refer to the point at the lower-left corner of the square as having coordinates *(1, 1)* and note that the sides of the square are orthogonal to each other and that the point at *(3, 1)* is *2* units from the point at *(1, 1)*. Now suppose that we remove the axes as shown in Figure 3.7. We can no longer specify where the points are. But those locations were relative to an arbitrary location of the origin and the orientation of the axes. What is more important is that the fundamental geometric relationships are preserved. The square is still a square, orthogonal lines are still orthogonal, and distances between points remain the same.

Of course, we may find it inconvenient, at best, to refer to a specific point as “that point over there” or “the blue point to the right of the red one”. Coordinate systems and frames solve this reference

problem, but for now we want to see just how far we can get following a coordinate free approach that does not require an arbitrary reference system.

3.1.3. THE MATHEMATICAL VIEW: VECTOR AND AFFINE SPACES

If we view scalars, points, and vectors as members of mathematical sets, then we can look at a variety of abstract spaces for representing and manipulating these sets of objects. Mathematicians have explored a variety of such spaces for applied problems, ranging from the solution of differential equations to the approximation of mathematical functions. The formal definitions of the spaces of interest to use – vector spaces, affine spaces, and Euclidean spaces – are given in Appendix B. We are concerned with only those examples in which the elements are geometric types.

We start with a set of scalars, any pair of which can be combined to form another scalar through two operations, called addition and multiplication. If these operations obey the closure, associativity, commutivity, and inverse properties described in Appendix B, the elements form a scalar field. Familiar examples of scalars include the real numbers, complex numbers, and rational functions.

Perhaps the most important mathematical space is the linear vector space. A vector space contains two distinct types of entities: vectors and scalars. In addition to the rules for combining scalars, within a vector space, we can combine scalars and vectors to form new vectors through scalar-vector multiplication and vectors with vectors through vector-vector addition. Examples of mathematical vector spaces include n-tuples of real numbers and the geometric operations on our directed line segments.

In a linear vector space, we do not necessarily have a way of measuring a scalar quantity. A Euclidean space is an extension of a vector space that adds a measure of size of distance and allows use to define such things as the length of a line segment.

An affine space is an extension of the vector space that includes an additional type of object: the point. Although there are no operations between two points or between a point and a scalar that yield points, there is an operation of vector-point addition that produces a new point. Alternately, we can say there is an operation called point-point subtraction that produces a vector from two points. Examples of affine spaces include the geometric operations on points and directed line segments that we introduced in Section 3.1.1.

In these abstract spaces, objects can be defined independently of any particular representation; they are simply members of various sets. One of the major vector space concepts is that of representing a vector in terms of one or more sets of basis vector. Representation provides the tie between abstract objects and their implementation. Conversion between representations leads us to geometric transformations.

3.1.4. THE COMPUTER SCIENCE VIEW

Although the mathematician may prefer to think of scalars, points, and vectors as members of sets that can be combined according to certain axioms, the computer scientist prefers to see them as abstract data types. An abstract data type is a set of operations on data; the operations are defined independently of how the data are represented internally or of how the operations are implemented. The notion of data

abstraction is fundamental to modern computer science. For example, the operation of adding an element to a list or multiplying two polynomials can be defined in dependently of how the list is stored or of how real numbers are represented on a particular computer. People familiar with this concept should have no trouble distinguishing between objects (and operations on objects) and objects' representations (or implementations) in a particular system. From a computational point of view, we should be able to declare geometric objects through code such as

```
vector u;
vector v;
point p;
point q;
scalar a;
scalar b;
```

regardless of the internal representation or implementation of the objects on a particular system. In object-oriented languages, such as c++, we can use language features, such as classes and overloading of operators, so we can write lines of code, such as

```
q = p + a * v;
```

using our geometric data types. Of course, first we must define functions that perform the necessary operations; so that we can write them, we must look at the mathematical functions that we wish to implement. First, we will define our objects. Then we will look to certain abstract mathematical spaces to help us with operations among them.

3.1.5. GEOMETRIC ABSTRACT DATA TYPE

The three views of scalars, points, and vectors leave us with a mathematical and computational framework for working with our geometric entities. In summary, for computer graphics our scalars are the real numbers using ordinary addition and multiplication. Our geometric points are locations in space, and our vectors are directed line segments. These objects obey the rules of an affine space. We can also create the corresponding abstract data types in a program.

Our next step is to show how we can our types to form geometrical objects and to perform geometric operations among them. We will use the following notation:

greek letters $\alpha, \beta, \gamma, \ldots$ denote scalars
uppercase letters $P, Q, R, \ldots$ denote points
lowercase letters $u, v, w, \ldots$ denote vectors

We have not as yet introduced any reference system, such as a coordinate system; thus, for vectors and points, this notation refers to the abstract objects, rather than to these objects' representations in a particular reference system. We use boldface letters for the latter in Section 3.3. The magnitude of a vector v is a real number denoted by $|v|$. The operation of vector-scalar multiplication has the property that

$$|\alpha v| = |\alpha||v|$$

and the direction of αv is the same as the direction of v if α is positive and the opposite direction if α is negative.

We have two equivalent operations that relate points and vectors. First, there is the subtraction of two points, P and Q – an operation that yields a vector v denoted by

$$v = P - Q$$

As a consequence of this operation, given any point Q and vector v, there is a unique point, P, that satisfies the preceding relationship. We can express this statement as follows: Given a point Q and a vector v, there a point P such that

$$P = Q + v$$

Figure 3.8. Point-point subtraction.

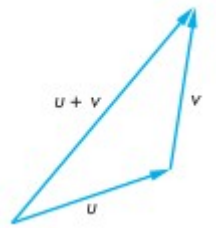

Figure 3.9.1. Use of the head-to-tail rule – for vectors.

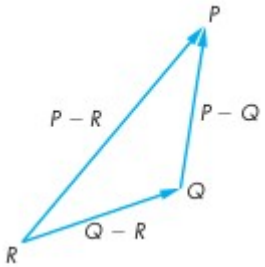

Figure 3.9.2. Use of the head-to-tail rule – for points.

Thus, P is formed by a point-vector addition operation. Figure 3.8 shows a visual interpretation of this operation. The head-to-tail gives us a convenient way of visualizing vector – vector addition. We obtain

the sum u + v as shown in Figure 3.9.1 by drawing the sum vector as connecting the tail of u to the head of v. However, we can also use this visualization, as demonstrated in Figure 3.9.2, to show that for any three points P, Q, and R,

$$(P-Q)+(Q-R)=P-R$$

3.1.6. LINES

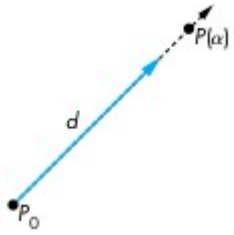

Figure 3.10. Line in a affine space.

The sum of a point and a vector (or the subtraction of two points) leads to the notion of a line in an affine space. Consider all points of the form

$$P(x)=P_o+xd$$

where P_0 is an arbitrary point, d is an arbitrary vector, and x is a scalar that can vary over some range of values. Given the rules for combining points, vectors, and scalars in an affine space, for any value of x, evaluation of the function $P(x)$ yields a point. For geometric vectors (directed line segments), these points lie on a line, as shown in Figure 3.10. This form is known as the parametric form of the line because we generate points on the line by varying the parameter x. For x = 0, the line passes through the point P_0, and as x is increased, all the points generated lie in the direction of the vector d. Thus, a line is infinitely long in both directions, a line segment is a finite piece of a line between two points, and a ray is infinitely long in one direction.

3.1.7. AFFINE SUMS

Whereas in an affine space the addition of two vectors, the multiplication of a vector by a scalar, and the addition of a vector and a point are defined, the addition of two arbitrary points and the multiplication of a point by a scalar are not. However, there is an operation called affine addition that has certain elements of these latter two operations. For any point Q, vector v, and positive scalar α,

$$P=Q+\alpha v$$

describes all points on the line from Q in the direction of v, as shown in Figure 3.11.

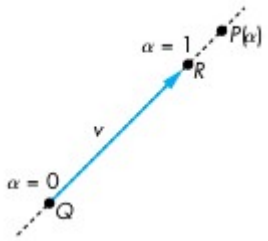

Figure 3.11. Affine addition.

However, we can always find a point R such that

$$v = R - Q$$

thus,

$$P = Q + \alpha(R - Q) = \alpha R + (1 - \alpha)Q$$

This operation looks like the addition of two points and leads to the equivalent form

$$P = \alpha_1 R + \alpha_2 Q$$

where

$$\alpha_1 + \alpha_2 = 1$$

3.1.8. CONVEXITY

A convex object is one for which any point lying on the line segment connecting any two points in the object is also in the object. We saw the importance of convexity for polygons in Chapter 2. We can use affine sums to help us gain a deeper understanding of convexity. For $0 \le \alpha \le 1$, the affine sum defines the line segment connecting R and Q, as shown 3.12; thus, this line segment is a convex object.

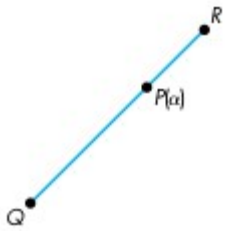

Figure 3.12. Line segment that connects two points.

We can extend the affine sum to include objects defined by n points $P_1, P_2, \ldots, P_n$. Consider the form

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n$$

We can show, by induction, that this sum is defined if and only if

$$\alpha_1+\alpha_2+\ldots+\alpha_n=1$$

The set of points formed by the affine sum of *n* points, under the additional restriction

$$\alpha_i \geq 0, i=1,2,\ldots,n$$

is called the convex hull of the set of points. It is easy to verify that the convex hull includes all line segments connecting pairs of points in $\{P_1, P_2, \ldots, P_n\}$. Geometrically, the convex hull is the set of points that we form by stretching a tight-fitting surface over the given set of points – shrink-wrapping the points. It is the smallest convex object that includes the set of points. The notion of convexity is extremely important in the design of curves and surfaces; we will return to it in Chapter 10.

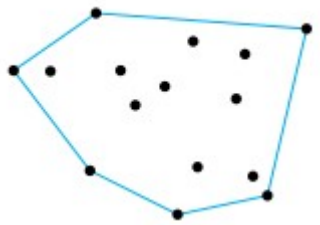

Figure 3.13. Convex hull.

3.1.9. DOT AND CROSS PRODUCTS

Many of the geometric concepts relating the orientation between two vectors are in terms of the dot (inner) and cross (outer) products of two vectors. The dot product of u and v is written $u \cdot v = 0$, and *u* and *v* are said to be orthogonal. In a Euclidean space, the magnitude of a vector is defined. The square of the magnitude of a vector is given by the dot product

$$|u|^2=u\cdot u$$

The cosine of the angle between two vectors is given by

$$\cos\theta=\frac{u\cdot v}{|u||v|}$$

In addition, $|u|\cos\theta = (u \cdot v) / |v|$ is the length of the orthogonal projection of *u* onto *v*, as shown in Figure 3.14. Thus, the dot product expresses the geometric result that the shortest distance from a point (the end of the vector *v*) to the line segment *v* is obtained by drawing the vector orthogonal to *v* from the end of *u*. We can also see that the vector *u* is composed of the vector sum of the orthogonal projection of *u* on *v* and a vector orthogonal to *v*.

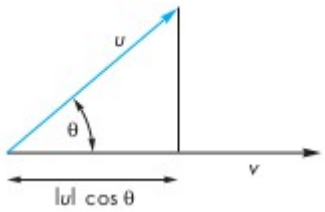

Figure 3.14. Dot product and project.

In a vector space, a set of vectors is linearly independent if we cannot write one of the vectors in terms of the others using scalar-vector addition. A vector space has a dimension, which is the maximum number of linearly independent vectors that we can find. Given any three linearly independent vectors in a three-dimensional space, we can use the dot product to construct three vectors, each of which is orthogonal to the other two. This process is outlined in Appendix B. We can also use two non-parallel vectors, *u* and *v*, to determine a third vector n that is orthogonal to them. This vector is the cross product

$$n = u \times v$$

Note that we can use the cross product to derive three mutually orthogonal vectors in a three dimensional space from any two non-parallel vectors. Starting again with *u* and *v*, we first compute *n* as before. Then, we can compute *w* by

$$w = u \times n$$

and *u*, *n*, and *w* are mutually orthogonal.

The cross product is derived in Appendix C, using the representation of the vectors that gives a direct method for computing it. The magnitude of the cross product gives the magnitude of the sine of the angle θ between *u* and *v*,

$$|\sin\theta| = \frac{|u \times v|}{|u||v|}$$

Note that the vectors *u*, *v*, and *n* form a right-handed coordinate system; that is, if *u* points in the direction of the thumb of the right hand and *v* points in the direction of the index finger, then *n* points in the direction of the middle finger.

3.1.10. PLANES

A plane in an affine space can be defined as a direct extension of the parametric line. From simple geometry, we know that three points not on the same line determine a unique plane. Suppose that *P*, *Q*, and *R* are three such points in an affine space. The line segment that joins *P* and *Q* is the set of points of the form

$$S(\alpha) = \alpha P + (1 - \alpha) Q, 0 \leq \alpha \leq 1$$

Suppose that we take an arbitrary point on this line segment and form the line segment from this point to *R*, as shown in Figure 3.16. Using a second parameter β, we can describe points along this line segment as

$$T(\beta)=\beta S+(1-\beta)R, 0\leq\beta\leq 1$$

Such points are determined by both α and β and form the plane determined by *P*, *Q*, and *R*. Combining the preceding two equations, we obtain one form of the equation of a plane:

$$T(\alpha,\beta)=\beta(\alpha P+(1-\alpha)Q)+(1-\beta)R$$

We can rearrange this equation in the following form:

$$T(\alpha,\beta)=P+\beta(1-\alpha)(Q-P)+(1-\beta)(R-P)$$

Noting that $Q - P$ and $R - P$ are arbitrary vectors, we have shown that a plane can also be expressed in terms of a point, P_0, and two non-parallel vectors, *u* and *v*, as

$$T(\alpha,\beta)=P_0+\alpha u+\beta v$$

If we write *T* as

$$T(\alpha,\beta)=\beta\alpha P+\beta(1-\alpha)Q+(1-\beta)R$$

this form is equivalent to expressing *T* as

$$T(\alpha,\beta',\gamma)=\alpha' P+\beta' Q+\gamma' R$$

as long as

$$\alpha'+\beta'+\gamma'=1$$

The representation of a point by $(\alpha', \beta', \gamma')$ is called its barycentric coordinate representation.

We can also observe that for $0 \leq \alpha, \beta \leq 1$, all the points $T(\alpha, \beta)$ lie in the triangle formed by *P*, *Q*, and *R*. If a point lies in the plane, then

$$P-P_0=\alpha u+\beta v$$

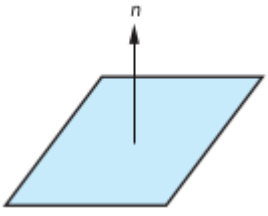

Figure 3.17. Normal to a plane.

We can find a vector *w* that is orthogonal to both *u* and *v*, as shown in Figure 3.17. If we use the cross product

$$n = u \times v$$

then the equation of the plane becomes

$$n \cdot P - P_0 = 0$$

The vector *n* is perpendicular, or orthogonal, to the plane; it is called the normal to the plane. The forms *P(α)*, for the line, and *T(α, β)*, for the plane, are known as parametric forms because they give the value of a point in space for each value to the parameters *α* and *β*.

3.2. THREE-DIMENSIONAL PRIMITIVES

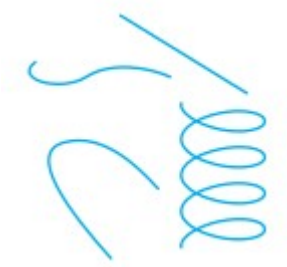

Figure 3.18. Curves in three dimensions.

Figure 3.19. Surfaces in three dimensions.

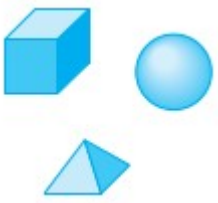

Figure 3.20. Volumetric objects.

In a three-dimensional world, we can have a far greater variety of geometric objects than we can in two dimensions. When we worked in a two dimensional plane in Chapter 2, we considered objects that were simple curves, such as line segments, and flat objects with well-defined interiors, such as simple polygons. In three dimensions, we retain these objects, but they are no longer restricted to lie in the

same plane. Hence, curves become curves in space, and objects with interiors can become surfaces in space. In addition, we can have objects with volumes, such as parallelepipeds and ellipsoids.

We face two problems when we expand our graphics system to incorporate all these possibilities. First, the mathematical definitions of these objects can become complex. Second, we are interested in only those objects that lead to efficient implementations in graphic systems. The full range of three-dimensional objects cannot be supported on existing graphic systems, except by approximate methods.

Three features characterize three dimensional objects that fit well with existing graphics hardware and software:

1. The objects are described by their surfaces and can be thought of as being hollow.
2. The objects can be specified through a set of vertices in three dimensions.
3. The objects either are composed of or can be approximated by flat, convex polygons.

We can understand why we set these conditions if we consider what most modern graphics system do best: They render triangles or meshes of triangles. Commodity graphics cards can render over 100 million small, flat triangles per second. Performance measurements for graphics system usually are quoted for small three dimensional triangles that can be generated by triangle strips. In addition, these triangles are shaded, lit, and texture mapped, features that are implemented in the hardware of modern graphic cards.

The first condition implies that we need only two dimensional primitives to model three dimensional objects because a surface is a two- rather than a three-dimensional entity. The second condition is an extension of our observations in Chapter 1 and 2. If an object is specified by vertices, we can use a pipeline architecture to process these vertices at high rate, and we can use the hardware to generate the images of the objects only during rasterization. The final condition is an extension from our discussion of two-dimensional polygons. Most graphics systems are optimized for the processing of points, line segments, and triangles. In three dimensions, a triangle is specified by an ordered list of three vertices.

However, for general polygons specified with more than three vertices, the vertices do not have to lie in the same plane. If they do not, there is no simple way to define the interior or the object. Consequently, most graphics system require that the application either specify simple planar polygons or triangles. If a system allows polygons and the application does not specify a flat polygon, then the result of rasterizing the polygon are not guaranteed to be what the programmer might desire. Because triangular polygons are always flat, either the modeling system is designed to always produce triangles, or the graphics system provides a method do divide, or tessellate, an arbitrary polygon into triangular polygons. If we apply this same argument to a curved object, such as a sphere, we realize that we should use an approximation to the sphere composed of small, flat polygons. Hence, even if our modeling system provides curved objects, we assume that a triangle mesh approximation is used for implementation.

The major exception to this approach is constructive solid geometry. In such systems, we build objects from a small set of volumetric objects through a set of operations such as union and intersection. We consider constructive solid geometry models in Chapter 8. Although this approach is an excellent one for modeling, rendering constructive solid geometry models is more difficult than is rendering surface-based polygon models. Although this situation may not hold in the future, we discuss in detail only surface rendering.

All the primitives with which we work can be specified through a set of vertices. As we move away from abstract objects to real objects, we must consider how we represent points in space in a manner that can be used within our graphics systems.

3.3. COORDINATE SYSTEMS AND FRAMES

So far, we have considered vectors and points as abstract objects, without representing them in an underlying reference system. In a three dimensional vector space, we can represent any vector w uniquely in terms of any three linearly independent vectors, v_1, v_2, and v_3, as

$$w=\alpha_1 v_1+\alpha_2 v_2+\alpha_3 v_3$$

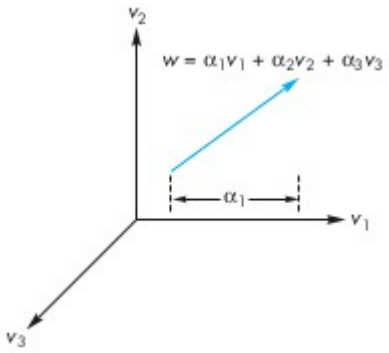

Figure 3.21. Vector derived from three basis vectors.

The scalar α_1, α_2, and α_3 are components of w with respect to the basis v_1, v_2, and v_3. These relationships are shown in Figure 3.21. We can write representation of w with respect to this basis as the column matrix

$$a=\begin{bmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\end{bmatrix}$$

where boldface letters denote a representation in a particular basis, as opposed to the original abstract vector w. We can also write this relationship as

$$w=\boldsymbol{a}^T\begin{bmatrix}v_1\\ v_2\\ v_3\end{bmatrix}=\boldsymbol{a}^T v$$

where

$$v=\begin{bmatrix}v_1\\ v_2\\ v_3\end{bmatrix}$$

We usually think of the basis vectors, v_1, v_2, v_3, as defining a coordinate system. However, for dealing with problems using points, vectors, and scalars, we need a more general method. Figure 3.22 show one aspect of the problem. The three vectors form a coordinate system that is shown in Figure 3.22.1 as we would usually draw it, with the three vectors emerging from a single point. We could use these three basis vectors as a basis to represent any vector in three dimensions. Vectors, however, have direction and magnitude but lack a position attribute. Hence 3.22.2 is equivalent, because we have moved the basis vectors, leaving their magnitudes and directions unchanged. Most people find this second figure confusing, even though mathematically it expresses the same information as the first figure. We are still left with the problem of how to represent points – entities that have fixed positions.

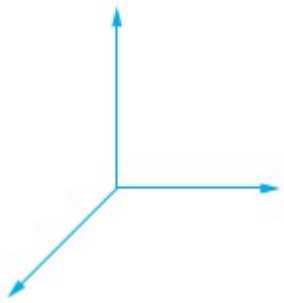

Figure 3.22.1. Coordinate systems – Vectors emerging from a common point.

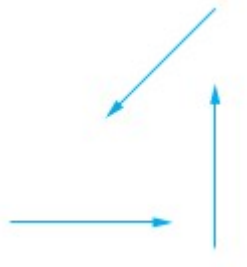

Figure 3.22.2. Coordinate systems – Vectors moved.

Because an affine space contains points, once we fix a particular reference point – the origin – in such a space, we can represent all points unambiguously. The usual convention for drawing coordinate axes as emerging from the origin, as shown in Figure 3.22.1, make sense in the affine space where both points and vectors have representations. However, this representation requires use to know both the reference point and the basis vectors. The origin and the basis vectors determine a frame. Loosely, this extension fixes the origin of the vector coordinate system at some point P_0. Within a given frame, every vector can be written uniquely as

$$w=\alpha_1 v_1+\alpha_2+v_2+\alpha_3 v_3=\boldsymbol{a}^T v$$

just as in a vector space; in addition, every point can be written uniquely as

$$P=P_0+\beta_1 v_1+\beta_2 v_2+\beta_3 v_3=P_0+\boldsymbol{b}^T v$$

Thus, the representation of a particular vector in a frame requires three scalars; the representation of a point requires three scalars and the knowledge of where the origin is located. As we will see in Section 3.3.4, by abandoning the more familiar notion of a coordinate system and a basis in that coordinate system in favor of the less familiar notion of a frame, we avoid the difficulties caused by vectors having magnitude and direction but no fixed position. In addition, we are able to represent points and vectors in a manner that will allow us to use matrix representations but that maintains a distinction between the two geometric types.

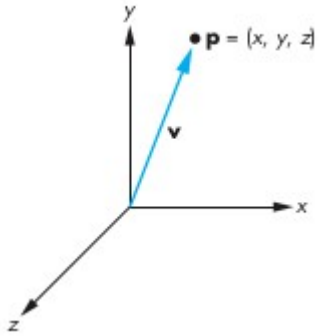

Figure 3.23. A dangerous representation of a vector.

Because points and vectors are two distinct geometric types, graphical representations that equate a point with a directed line segment drawn from the origin to that point should be regarded with suspicion. Thus, a correct interpretation of Figure 3.23 ins that given a vector can be defined as going from a fixed reference point (the origin) to a particular point in space. Note that a vector, like a point, exists regardless of the reference system, but as we will see with both points and vectors, eventually we have to work with their representation in a particular reference system.

3.3.1. REPRESENTATIONS AND N-TUPLES

Suppose that vectors e_1, e_2, and e_3 form a basis. The representation of any vector, v, is given by the component $(\alpha_1, \alpha_2, \alpha_3)$ of a vector $\boldsymbol{a}$ where

$$v=\alpha_1 e_1+\alpha_2 e_2+\alpha_3 e_3$$

The basis vectors must themselves have representations that we can denote $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, and $\boldsymbol{e_3}$, given by

$$\boldsymbol{e_1}=(1,0,0)^T$$
$$\boldsymbol{e_2}=(0,1,0)^T$$
$$\boldsymbol{e_3}=(0,0,1)^T$$

In other words, the 3 tuples $(1, 0, 0)$ is the representation of the first basis vector. Consequently, rather than thinking in terms of abstract vectors, we can work with 3 tuples and we can write the representation of any vector v as a column matrix $\boldsymbol{a}$ or the 3 tuple $(\alpha_1, \alpha_2, \alpha_3)$, where

$$\boldsymbol{a}=\alpha_1 \boldsymbol{e_1}+\alpha_2 \boldsymbol{e_2}+\alpha_3 \boldsymbol{e_3}$$

The basis 3 tuples $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, and $\boldsymbol{e_3}$ are vectors in the familiar Euclidean space $\boldsymbol{R^3}$. The vector space $\boldsymbol{R^3}$ is equivalent (or homomorphic) to the vector space of our original geometric vectors. From a practical perspective, it is always easier to work with 3 tuples (or more generally n tuples) than with other representations.

3.3.2. CHANGE OF COORDINATE SYSTEMS